链滴

Google Guice 入门示例

作者：flowaters

原文链接：https://ld246.com/article/1498879285223

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

Google Guice是一个依赖注入(DI)框架，实现了控制反转(IOC)。

优点：比spring轻，被越来越多的开源工具使用

缺点：调用时出现异常时，Guice不会抛出异常。需要开发者自己解决。

本文用一个示例，来说明这种调用。

<h2>
背景
</h2>

在数据流的处理中，会用到生产者消费者模型，如通过http服务接收到日志数据，然后经过业务处，最终存储到本地磁盘中。

随着业务变化，数据生产者会变化，如不再通过http接收数据，而是通过从消息队列中读取数据。据消费者也会变化，如不再存储到本地磁盘，而是存储到远程的Hadoop中。

在这个过程中，业务处理逻辑是不变的，我们也希望能够复用业务代码逻辑，同时可以选用合适的产和消费实现。

<h2>
传统实现
</h2>

那么怎么做呢？一般会写三个接口文件：生产者，消费者，业务处理者，如下：

// 业务处理接口


```
<pre class="prettyprint lang-js">public interface GatewayService {


void start();

void stop();

}
</pre>

<pre class="prettyprint lang-java">// 生产者接口</pre>

<pre class="prettyprint lang-java">public interface ProduceService {

    List&lt;String&gt; getDataList();

    void start();

    void stop();
}
</pre>

<p>
    <br />
</p>
<p>
    <br />
</p>
<pre class="prettyprint lang-js">// 消费者接口</pre>
<pre class="prettyprint lang-js">public interface ConsumeService {


void consume(List&lt;String&gt; dataList);

}</pre>

<p>
    <br />
</p>
<p>
    然后分别实现：
</p>
<p>
    <br />
</p>
<pre class="prettyprint lang-java">

<pre class="prettyprint lang-java">// 业务处理实现类</pre>

<pre class="prettyprint lang-java">public class GatewayServiceImpl implements GatewayServ
ce {
```

```
    private ConsumeService consumer;

    private ProduceService producer;

    private volatile boolean isRunning = false;

    public GatewayServiceImpl(ProduceService producer, ConsumeService consumer) {
        this.producer = producer;
        this.consumer = consumer;
    }

    public void start() {
        if (isRunning) {
            throw new IllegalStateException("already running");
        }

        new Thread(new Runnable() {
            public void run() {
                try {
                    isRunning = true;
                    while (isRunning) {
                        List&lt;String&gt; dataList = producer.getDataList();
                        consumer.consume(dataList);
                    }
                } finally {
                    isRunning = false;
                }
            }
        }, "gateway-service").start();
    }

    public void stop() {
        this.isRunning = false;
    }
}
</pre>

<br />
</pre>
<pre class="prettyprint lang-java">// 生产者基类</pre>
<p>
    <br />
</p>
<p>
    <br />
</p>
<pre class="prettyprint lang-java">public abstract class AbstractProducerService implements
ProduceService {


public void start() {
}
```

```
public void stop() {
}

}
```

</pre>

<pre class="prettyprint lang-java">// 生产类实现类</pre>
<pre class="prettyprint lang-java">public class MockProducer extends AbstractProducerServi
e implements ProduceService {

private static AtomicLong counter = new AtomicLong(0);

public List<String> getDataList() {
    try {
        Thread.sleep(500L);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return Collections.singletonList(String.valueOf(counter.incrementAndGet()));
}

}
</pre>

<pre class="prettyprint lang-js">// 消费者实现类</pre>

<pre class="prettyprint lang-js">public class MockConsumer implements ConsumeService {

    public void consume(List<String> dataList) {
        System.out.println("MockConsumer: " + dataList);
    }
}
</pre>

  最后把代码组装起来

<pre class="prettyprint lang-java">

<pre class="prettyprint lang-java">   public static void main(String[] args) throws Interrupted
xception {
        ProduceService producer = new MockProducer();
        ConsumeService consumer = new MockConsumer();
        GatewayService service = new GatewayServiceImpl(producer, consumer);
        service.start();
        Thread.sleep(10000L);
        service.stop();
    }</pre>

</pre>

    这是一般的写法

<h2>
    Guice实现
</h2>

    如果采用Guice的话，将new 新的实例的过程，由guice来接管。

<pre class="prettyprint lang-java">public class MockGatewayModule implements Module {

public void configure(Binder binder) {
    binder.bind(ConsumeService.class).to(MockConsumer.class);
    binder.bind(ProduceService.class).to(MockProducer.class);
    binder.bind(GatewayService.class).to(GatewayServiceImpl.class).asEagerSingleton();
}

}</pre>

    然后将GatewayService接口的初始化中参数的传入，由注入来实现，下面的代码中，增加了一个 @
nject注解

<pre class="prettyprint lang-java">public class GatewayServiceImpl implements GatewayServ
ce {

private ConsumeService consumer;

private ProduceService producer;

private volatile boolean isRunning = false;

@Inject
public GatewayServiceImpl(ProduceService producer, ConsumeService consumer) {
    this.producer = producer;
    this.consumer = consumer;
}

public void start() {
    if (isRunning) {
        throw new IllegalStateException("already running");
    }

    new Thread(new Runnable() {
        public void run() {
            try {
                isRunning = true;
                while (isRunning) {
                    List<String> dataList = producer.getDataList();
                    consumer.consume(dataList);
                }
            } finally {
                isRunning = false;
            }
        }
    }, "gateway-service").start();
}

public void stop() {
    this.isRunning = false;
}

}</pre>

  最近新的代码组装方式：

<pre class="prettyprint lang-java">   public static void main(String[] args) throws Interrupted
xception {
        Injector injector = Guice.createInjector(new MockGatewayModule());
        injector.getInstance(GatewayService.class).start();
        Thread.sleep(10000L);
        injector.getInstance(GatewayService.class).stop();
    }</pre>
<h2>
  结果
</h2>

  执行效果是相同的，但是有下面的两个特点：

    接口实例的产生放在了Module中来进行，省去了new的过程

    通过注入，避免了实例中参数的显式传入

  进一步阅读：

      Motivat
on

        <a href="http://blog.csdn.net/zhaoruixiang1111/article/details/52852249" target="
blank">Guice基本用法

```
        </li>
        <li>
          <a href="http://superleo.iteye.com/blog/314816" target="_blank">Guice2.0的变化
—第一部分 新的特性（上）</a><br />
        </li>
      </ul>
    </p>
</p>
<p>
    <br />
</p>
<p>
    <br />
</p>
<p>
    <br />
</p>
<p>
    <br />
</p>
<p>
    <br />
</p>
```

原文链接：Google Guice 入门示例